mathematics

MDPI

Article

Accurate Approximation of the Matrix Hyperbolic Cosine Using Bernoulli Polynomials

José M. Alonso [1,*], Javier Ibáñez [2], Emilio Defez [2] and Fernando Alvarruiz [3]

[1] Instituto de Instrumentación para Imagen Molecular, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain
[2] Instituto de Matemática Multidisciplinar, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain
[3] Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain
* Correspondence: jmalonso@dsic.upv.es

Abstract: This paper presents three different alternatives to evaluate the matrix hyperbolic cosine using Bernoulli matrix polynomials, comparing them from the point of view of accuracy and computational complexity. The first two alternatives are derived from two different Bernoulli series expansions of the matrix hyperbolic cosine, while the third one is based on the approximation of the matrix exponential by means of Bernoulli matrix polynomials. We carry out an analysis of the absolute and relative forward errors incurred in the approximations, deriving corresponding suitable values for the matrix polynomial degree and the scaling factor to be used. Finally, we use a comprehensive matrix testbed to perform a thorough comparison of the alternative approximations, also taking into account other current state-of-the-art approaches. The most accurate and efficient options are identified as results.

Keywords: Bernoulli matrix polynomials; matrix hyperbolic cosine; matrix functions approximation

MSC: 65F60

check for updates

Citation: Alonso, J.M.; Ibáñez, J.; Defez, E.; Alvarruiz, F. Accurate Approximation of the Matrix Hyperbolic Cosine Using Bernoulli Polynomials. *Mathematics* **2023**, *11*, 520. https://doi.org/10.3390/math11030520

Academic Editor: Sitnik Sergey

Received: 12 December 2022
Revised: 13 January 2023
Accepted: 16 January 2023
Published: 18 January 2023

1. Introduction

Due to their use in many engineering and scientific applications, the numerical computation of matrix functions has received remarkable and growing attention in recent years. For instance, the efficient evaluation of matrix functions is part of reduced-order models [1] and (pp. 275–303 in [2]), image denoising [3] and graph neural networks [4], among other applications.

The field of approximation theory for matrix functions is quite extensive (see, e.g., the corresponding chapters of [5]). The best-known methods are based on rational or polynomial approximations, or on different matrix decomposition techniques (e.g., Schur).

Among the different matrix functions, the hyperbolic ones must be highlighted. A set of state-of-the-art algorithms developed by the authors to calculate matrix hyperbolic sine and cosine functions can be found in [6–10]. Very recent generalizations of these matrix functions can also be found in references [11,12]. Among many others fields, hyperbolic sine and cosine functions are applied in the study of communicability analysis in complex networks [13,14] or to construct the exact series solution of coupled hyperbolic systems [15,16]. Additionally, algorithms for the matrix inverse hyperbolic cosine and sine are included in [17], while methods for computing the action of the hyperbolic cosine or sine of a matrix on a vector are provided in [18,19].

On the other hand, different numerical methods have been recently proposed for the effective calculation of the matrix hyperbolic tangent function [20]. This matrix function is used, for example, to give an analytical solution of the radiative transfer equation [21],

in the heat transference field [22,23], in the study of symplectic systems [24,25], in graph theory [26] and in the development of special types of exponential integrators [27,28].

In addition, the generalizations of some well-known classical special functions into matrix frameworks are important both from the theoretical and applied points of view. These new extensions (Laguerre, Hermite, Chebyshev, Jacobi matrix polynomials, etc.) have proved to be very useful in various fields, such as physics, engineering, statistics and telecommunications. Recently, Bernoulli polynomials $B_n(x)$, which are defined in [29] as the coefficients of the generating function

$$g(x,t) = \frac{te^{tx}}{e^t - 1} = \sum_{n\geq 0} \frac{B_n(x)}{n!} t^n \ , \ |t| < 2\pi, \tag{1}$$

and which have the explicit expression for $B_n(x)$

$$B_n(x) = \sum_{k=0}^{n} \binom{n}{k} \mathcal{B}_k x^{n-k}, \tag{2}$$

where the *Bernoulli numbers* are defined by $\mathcal{B}_n = B_n(0)$, satisfying the explicit recurrence

$$\mathcal{B}_0 = 1, \qquad \mathcal{B}_n = -\sum_{k=0}^{n-1} \binom{n}{k} \frac{\mathcal{B}_k}{n+1-k}, \ n \geq 1, \tag{3}$$

were generalized to the matrix framework in [30]. Excluding $\mathcal{B}_1 = -0.5$, all Bernoulli numbers $\mathcal{B}_n$, with n being an odd number, are null (see Appendix A, Remark A1, for the deduction of formula (3)).

For a matrix $A \in \mathbb{C}^{r\times r}$, the n-th Bernoulli matrix polynomial is defined by the expression

$$B_n(A) = \sum_{k=0}^{n} \binom{n}{k} \mathcal{B}_k A^{n-k}. \tag{4}$$

For these matrix polynomials, we have the following series expansion of the matrix exponential function:

$$e^{At} = \left(\frac{e^t - 1}{t}\right) \sum_{n\geq 0} \frac{B_n(A)t^n}{n!} \ , \ |t| < 2\pi. \tag{5}$$

To obtain *practical* approximations of the matrix exponential function using expression (5), we use the scaling and squaring technique [31,32]. This method is based on the well-known property of

$$e^A = \left(e^{A2^{-s}}\right)^{2^s}, \tag{6}$$

where $s \geq 0$ is an integer, called the scaling factor, to be determined in order to reduce the norm of matrix A appropriately. Let us take m as the approximation polynomial degree to be used. Then, from (5), using $t = 1$, we have

$$e^{A2^{-s}} \approx (e-1) \sum_{n=0}^{m} \frac{B_n(A2^{-s})}{n!}. \tag{7}$$

Once approximation (7) is computed, s squaring steps must be carried out to reverse the scaling effect to finally obtain e^A. As an objective of this work, an algorithm (described in Section 2) was developed to determine the most appropriate values of m and s.

The use of expansion (5) to approximate the matrix exponential function with good results of precision and computational cost can be found in [30]. For a matrix $A \in \mathbb{C}^{r\times r}$, using expression (5), we obtain (see Appendix A, Remark A2)

$$\cosh(A) = \sinh(1) \sum_{n\geq 0} \frac{B_{2n}(A)}{(2n)!} + (\cosh(1) - 1) \sum_{n\geq 0} \frac{B_{2n+1}(A)}{(2n+1)!}. \tag{8}$$

Notice that unlike what happens when considering hyperbolic cosine series expansions using Taylor or Hermite polynomials, all Bernoulli polynomials are needed in the development of $\cosh(A)$ (and not just the even-numbered). However, this also possible by operating to obtain an alternative approximation to the matrix hyperbolic cosine where only polynomials of even degree appear, as follows (see Appendix A, Remark A3):

$$\cosh(A) = \sinh(1) \sum_{n \geq 0} \frac{2^{2n} B_{2n}\left(\frac{1}{2}(A+I)\right)}{(2n)!}. \tag{9}$$

Currently, few methods addressing the effective computation of the matrix hyperbolic cosine for matrices of non-trivial size are available in the literature, such as those appearing, e.g., in graph theory [14]. For instance, the work [33] (which presents the available software, according to the authors, for the computation of matrix functions) indicates only two codes for the computation of the matrix hyperbolic cosine (MATLAB's `funm` function, based on the Schur–Parlett algorithm for general functions [34], and `thfm`, included in GNU Octave's extra package `linear-algebra` and based on the computation of the matrix exponential function by means of `expm`).

Therefore, the main objective, and the novelty, of this work is to address that need for methods by designing, implementing and evaluating different algorithms for the computation of the matrix hyperbolic cosine by means of Bernoulli polynomials. Some of the methods proposed are based on approximations (8) and (9), and others are based on the computation of the matrix exponential using Bernoulli polynomials.

Hereafter, we denote, with $\mathbb{C}^{r\times r}$, the set of all complex square matrices of order r, as mentioned above, and with I, the identity matrix. Additionally, a matrix polynomial of degree m, for $A \in \mathbb{C}^{r\times r}$, is given by the expression $P_m(A) = p_0 I + p_1 A + \cdots + p_{m-1}A^{m-1} + p_m A^m$, where coefficients $p_i, 0 \leq i \leq m$, are complex numbers. The result of rounding a real number x to the nearest integer greater than or equal to x is denoted with $\lceil x \rceil$. Additionally, the result of rounding x to the nearest integer less than or equal to x is represented by $\lfloor x \rfloor$. Finally, matrix norm $\|\cdot\|$ represents any subordinate matrix norm. In particular, $\|\cdot\|_2$ stands for the traditional 2-norm.

The paper is organized as follows: In Section 2, three algorithms that compute the matrix hyperbolic cosine function are described. They are based either on the previous Bernoulli series expansions or on the theoretical definition of the matrix hyperbolic cosine in terms of the matrix exponential, which also derives from its series expansion based on Bernoulli polynomials. After the appropriate implementation of all these algorithms in their respective MATLAB codes, Section 3 presents an exhaustive comparison among all of them in order to choose the most appropriate one. For this purpose, numerous experiments were carried out, where their numerical and computational performance were evaluated against a widely heterogeneous testbed composed of three types of matrices. Finally, conclusions are given in Section 4.

2. The Proposed Algorithms

2.1. Algorithms Based on the Bernoulli Series of the Matrix Hyperbolic Cosine

By truncating series (8), one obtains the $m-$th order Bernoulli approximation to the matrix hyperbolic cosine (we assume that m is even for simplicity of exposition)

$$\cosh(A) \approx P_m(A) = \sinh(1) \sum_{n=0}^{m/2} \frac{B_{2n}(A)}{(2n)!} + (\cosh(1) - 1) \sum_{n=0}^{m/2-1} \frac{B_{2n+1}(A)}{(2n+1)!}, \tag{10}$$

where polynomial P_m can be expressed as

$$P_m(A) = \sum_{k=0}^{m} p_k^{(m)} A^k.$$

Similarly, by truncating series (9), the second alternative $m-$th order Bernoulli approximation to our target matrix function is obtained as

$$\cosh(A) \approx \sinh(1) \sum_{n=0}^{m/2} \frac{2^{2n} B_{2n}\left(\frac{1}{2}(A+I)\right)}{(2n)!}, \tag{11}$$

which is a polynomial of order m with all odd-order terms being equal to zero. Thus, by defining $\bar{A} := A^2$, we obtain a polynomial $\bar{P}_{\bar{m}}$ of order $\bar{m} = m/2$.

$$\cosh(A) \approx \bar{P}_{\bar{m}}(\bar{A}) = \sum_{k=0}^{\bar{m}} \bar{p}_k^{(\bar{m})} \bar{A}^k, \tag{12}$$

where $p_k^{(m)}$ and $\bar{p}_k^{(\bar{m})}$ are coefficients dependent on order m and on the truncated series considered, respectively. These Bernoulli polynomial coefficients become more and more similar to those of the Taylor series as degree m of the polynomial increases. In the case of the polynomial from (12), the coefficients converge to the even-order coefficients of the Taylor series (with the odd-order coefficients being zero).

Algorithms 1 and 2 are related to the hyperbolic cosine computation of a matrix A using formulation (10) or (11), where the scaling and squaring technique is considered. In Line 1 of Algorithm 1 (Line 2 of Algorithm 2), the most appropriate values corresponding to order $m = m_k$ of the approximation polynomial and scaling parameter s are found out, attempting to reduce the norm of matrix A and calculate $\cosh(A)$ as accurately as possible. That will be discussed in Section 2.3.

Next, in Line 2 of Algorithm 1 (Line 3 of Algorithm 2), matrix A is properly scaled, and in the following line, polynomial $P_{m_k}(A)$ or $\bar{P}_{m_k}(A)$ must be efficiently computed using methods such as those described in [35,36]. In our implementations, the Paterson–Stockmeyer method [35] was employed. In this procedure, assuming that polynomial order m_k is chosen from the set

$$\mathbb{M} = \{m_1, m_2, \dots\} = \{1, 2, 4, 6, 9, 12, 16, 20, 25, 30, 36, 42, 49, 56, 64, \dots\},$$

powers A^i, $2 \le i \le q$, must be calculated, where $q = \lceil \sqrt{m_k} \rceil$ or $q = \lfloor \sqrt{m_k} \rfloor$ is an integer divisor of m_k. With these matrix powers A^i, we can efficiently compute $P_{m_k}(A)$ as

$$\begin{aligned} &P_{m_k}(A) = \\ &(((p_{m_k} A^q + p_{m_k-1} A^{q-1} + p_{m_k-2} A^{q-2} + \cdots + p_{m_k-q+1} A + p_{m_k-q} I) A^q \\ &\quad + p_{m_k-q-1} A^{q-1} + p_{m_k-q-2} A^{q-2} + \cdots + p_{m_k-2q+1} A + p_{m_k-2q} I) A^q \\ &\quad + p_{m_k-2q-1} A^{q-1} + p_{m_k-2q-2} A^{q-2} + \cdots + p_{m_k-3q+1} A + p_{m_k-3q} I) A^q \\ &\quad \cdots \\ &\quad + p_{q-1} A^{q-1} + p_{q-2} A^{q-2} + \cdots + p_1 A + p_0 I, \end{aligned} \tag{13}$$

where k matrix products are involved.

Finally, in Lines 4–6 of Algorithm 1 (Lines 5–7 of Algorithm 2), $\cosh(A)$ is appropriately recovered by repeatedly using the double-angle formula $\cosh(2A) = 2\cosh^2(A) - I$.

Algorithm 1: Given a matrix $A \in \mathbb{C}^{r\times r}$, a minimum order $m_{lower} \in \mathbb{M}$ and a maximum order $m_{upper} \in \mathbb{M}$, this algorithm computes $C = \cosh(A)$ with the Bernoulli series (10)

1 Select suitable values of $m_k \in \mathbb{M}$, $m_{lower} \leq m_k \leq m_{upper}$, and $s \in \mathbb{N} \cup \{0\}$ for the Bernoulli approximation (10) of $\cosh(2^{-s}A)$ (see Section 2.3)
2 $A = 2^{-s}A$
3 $C = P_{m_k}(A)$ `/* Compute` $P_{m_k}(A)$ `in (10) by (13) */`
4 **for** $i = 1$ **to** s **do** `/* Recover` $\cosh(A)$ `*/`
5 $\quad C = 2C^2 - I$
6 **end**

Algorithm 2: Given a matrix $A \in \mathbb{C}^{r\times r}$, a minimum order $m_{lower} \in \mathbb{M}$ and a maximum order $m_{upper} \in \mathbb{M}$, this algorithm computes $C = \cosh(A)$ with the Bernoulli series (11)

1 $\bar{A} = A^2$
2 Select suitable values of $m_k \in \mathbb{M}$, $m_{lower} \leq m_k \leq m_{upper}$, and $s \in \mathbb{N} \cup \{0\}$, to approximate $\cosh(2^{-s}A)$ using $\bar{P}_m(4^{-s}\bar{A})$ (see Section 2.3)
3 $A = 4^{-s}\bar{A}$
4 $C = \bar{P}_{m_k}(A)$ `/* Compute` $\bar{P}_{m_k}(A)$ `in (12) by (13) */`
5 **for** $i = 1$ **to** s **do** `/* Recover` $\cosh(A)$ `*/`
6 $\quad C = 2C^2 - I$
7 **end**

2.2. Algorithm Based on the Bernoulli Series of the Matrix Exponential

Another way to approximate the matrix hyperbolic cosine is by means of Algorithm 3, which uses the formula

$$\cosh(A) = \frac{e^A + e^{-A}}{2}, \tag{14}$$

and computes the matrix exponential by means of ([30] Algorithm 1), although in this paper, we use forward error analysis, as in the previous section, instead of the backward error used in [30].

Algorithm 3: Given a matrix $A \in \mathbb{C}^{r\times r}$, a minimum order $m_{lower} \in \mathbb{M}$ and a maximum order $m_{upper} \in \mathbb{M}$, this algorithm computes $C = \cosh(A)$ with the Bernoulli series of the matrix exponential using the formula $\cosh(A) = \frac{e^A+e^{-A}}{2}$

1 Select suitable values of $m_k \in \mathbb{M}$, $m_{lower} \leq m_k \leq m_{upper}$, and $s \in \mathbb{N} \cup \{0\}$ for the Bernoulli approximation of $e^{2^{-s}A}$ (see Section 2.3)
2 $A = 2^{-s}A$
3 $E_1 = \tilde{P}_{m_k}(A)$ and $E_2 = \tilde{P}_{m_k}(-A)$ by using (8) from [30] and (13)
4 **for** $i = 1$ **to** s **do** `/* Recover` e^A `and` e^{-A} `*/`
5 $\quad E_1 = E_1^2$
6 $\quad E_2 = E_2^2$
7 **end**
8 $C = \frac{E_1+E_2}{2}$

In Line 1, Algorithm 3 selects the suitable values of $m_k \in \mathbb{M}$ and $s \in \mathbb{N} \cup \{0\}$, according to Section 2.3, for computing the Bernoulli approximation to the matrix exponential using the scaling and squaring procedure. Once the norm of matrix A is reduced in Line 2 of Algorithm 3, then $E_1 = \tilde{P}_{m_k}(2^{-s}A)$ and $E_2 = \tilde{P}_{m_k}(-2^{-s}A)$ are calculated in Line 3, where $\tilde{P}_{m_k}$ is a polynomial approximation to the matrix exponential function by means of Bernoulli matrix polynomials, according to (8) from [30]. The evaluation of $\tilde{P}_{m_k}$ is performed using

the Paterson–Stockmeyer method (13). Next, in Lines 4–7 of Algorithm 3, e^A and e^{-A} are recovered. Finally, $\cosh(A)$ is computed in Line 8.

2.3. Selecting the Order of Polynomials and the Scaling Factor

The computation of scaling factor s and order m of the Bernoulli approximation in the previous three algorithms is based on the absolute or relative forward error, presented next. We first consider approximation polynomial P_m derived from (10). Let m be a large enough value such that coefficients $p_i^{(m)}$ of Bernoulli approximation $P_m(A)$ from (10) to $\cosh(A)$ are practically identical to those of the Taylor approximation. Then, the absolute forward error when computing $P_m(A)$ can be calculated and bounded as follows:

$$E_{af}(P_m(A)) = \|\cosh(A) - P_m(A)\| \approx \left\| \sum_{i>m} a_i A^i \right\|, \tag{15}$$

where $\sum_{i>m} a_i A^i$ is the absolute forward error series of the Taylor approximation of order m.

Let $h_m(x) = \sum_{i>m} a_i x^i$ and $\tilde{h}_m(x) = \sum_{i>m} |a_i| x^i$. If Theorem 1.1 from [37] is applied, then

$$E_{af}(P_m(A)) \approx \|h_m(A)\| \leq \tilde{h}_m(\alpha_m),$$

where

$$\alpha_m = \max\left\{ \left\|A^i\right\|^{1/i} : \ i = m+1, m+2, \cdots, 2m+1 \right\}.$$

Let

$$\Theta_m = \max\left\{ \theta \geq 0 : \sum_{i>m} |a_i| \theta^i \leq u \right\}, \tag{16}$$

where $u = 2^{-53}$ is the unit roundoff in IEEE double-precision arithmetic.

If

$$\alpha_m < \Theta_m, \tag{17}$$

then we have

$$E_{af}(P_m(A)) \approx \|h_m(A)\| \leqslant \tilde{h}_m(\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u. \tag{18}$$

and scaling parameter s is 0.

However, if (17) is not fulfilled, then the smallest value of s, such that $2^{-s}\alpha_m < \Theta_m$, must be determined. In this case, we obtain

$$E_{af}(P_m(2^{-s}A)) \approx \left\|h_m(2^{-s}A)\right\| \leqslant \tilde{h}_m(2^{-s}\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u. \tag{19}$$

On the other hand, if $\cosh(A)$ is also invertible and, once again, m is a sufficiently large value such that terms $p_i^{(m)}$ of Bernoulli approximation $P_m(A)$ to our goal matrix function are equivalent to those of the Taylor one, then the relative forward error corresponding to this approximation can be computed and bounded in the following way:

$$\begin{aligned} E_{rf}(P_m(A)) &= \left\|\cosh(A)^{-1}(\cosh(A) - P_m(A))\right\| = \\ &= \left\|I - \cosh(A)^{-1} P_m(A)\right\| \approx \left\| \sum_{i>m} b_i A^i \right\|, \end{aligned}$$

where $\sum_{i>m} b_i A^i$ is the relative forward error series of the Taylor approximation of order m.

Similarly to the case of the absolute forward error, let $g_m(x) = \sum_{i>m} b_i x^i$ and $\tilde{g}_m(x) = \sum_{i>m} |b_i| x^i$. Then, from ([37] Theorem 1.1), we have

$$E_{rf}(P_m(A)) \approx \|g_m(A)\| \leq \tilde{g}_m(\alpha_m).$$

Let $\hat{\Theta}_m$ be

$$\hat{\Theta}_m = \max\left\{\theta \geq 0 : \sum_{i>m} |b_i|\theta^i \leq u\right\}. \tag{20}$$

If the condition

$$\alpha_m < \hat{\Theta}_m, \tag{21}$$

holds, then we have

$$E_{rf}(P_m(A)) \approx \|g_m(A)\| \leqslant \tilde{g}_m(\alpha_m) \leqslant \tilde{g}_m(\hat{\Theta}_m) \leqslant u. \tag{22}$$

Conversely, if (21) is not verified, then the smallest value of s, such that $2^{-s}\alpha_m < \hat{\Theta}_m$, is established. In this case, we obtain

$$E_{rf}(P_m(2^{-s}A)) \approx \left\|g_m(2^{-s}A)\right\| \leqslant \tilde{g}_m(2^{-s}\alpha_m) \leqslant \tilde{g}_m(\hat{\Theta}_m) \leqslant u. \tag{23}$$

The values of Θ_{m_k} and $\hat{\Theta}_{m_k}$, $m_k \in \mathbb{M}$, for $E_{af}(P_{m_k}(A))$ and $E_{rf}(P_{m_k}(A))$, respectively, were computed using MATLAB Symbolic Math Toolbox. All of them appear in Table 1.

An analogous study was carried out for Bernoulli approximation $\cosh(A) \approx \bar{P}_{\bar{m}}(\bar{A})$ from (12). In this case, we assume that $\bar{m}$ is a large enough value such that coefficients $\bar{p}_k^{(\bar{m})}$ of polynomial $\bar{P}_{\bar{m}}$ are practically identical to the corresponding even-order coefficients of the Taylor approximation to $\cosh(A)$, while the odd-order coefficients are zero. Then, the absolute forward error is

$$E_{af}(\bar{P}_{\bar{m}}(\bar{A})) = \left\|\cosh(A) - \bar{P}_{\bar{m}}(\bar{A})\right\| \approx \left\|\sum_{i>2\bar{m}} a_i A^i\right\| = \left\|\sum_{i>\bar{m}} \bar{a}_i \bar{A}^i\right\|,$$

where $\bar{A} = A^2$, coefficients a_i are from (15) and $\bar{a}_i = a_{2i}$. Similarly, the relative forward error is

$$E_{rf}(\bar{P}_{\bar{m}}(\bar{A})) = \left\|\cosh(A)^{-1}(\cosh(A) - \bar{P}_{\bar{m}}(\bar{A}))\right\| \approx \left\|\sum_{i>\bar{m}} b_{2i}\bar{A}^i\right\| \approx \left\|\sum_{i>\bar{m}} \bar{b}_i \bar{A}^i\right\|,$$

where $\bar{b}_i = b_{2i}$.

Let Θ_{P_m} and $\hat{\Theta}_{P_m}$ be the values of Θ_m and $\hat{\Theta}_m$ for polynomial P_m as defined in (16) and (20), respectively. We can analogously define the corresponding values for polynomial $\bar{P}_{\bar{m}}$, $\Theta_{\bar{P}_{\bar{m}}}$ and $\hat{\Theta}_{\bar{P}_{\bar{m}}}$, and it is easy to see that $\Theta_{\bar{P}_{\bar{m}}} = \Theta^2_{P_{2\bar{m}}}$, $\hat{\Theta}_{\bar{P}_{\bar{m}}} = \hat{\Theta}^2_{P_{2\bar{m}}}$.

Table 1. Values of Θ_{m_k} and $\hat{\Theta}_{m_k}$, $m_k \in \mathbb{M}$, for $E_{af}(P_{m_k}(A))$ and $E_{rf}(P_{m_k}(A))$, respectively.

m_k	Θ_{m_k}	$\hat{\Theta}_{m_k}$
1	$1.4901161193847656 \times 10^{-8}$	$1.4901161193847656 \times 10^{-8}$
2	$2.2719845183149197 \times 10^{-4}$	$2.2719845056098161 \times 10^{-4}$
4	$6.5633223103254337 \times 10^{-3}$	$6.5633004324626544 \times 10^{-3}$
6	$3.8138663224761025 \times 10^{-2}$	$3.8135350033771671 \times 10^{-2}$
9	$1.1495105955344324 \times 10^{-1}$	$1.1487736634745561 \times 10^{-1}$
12	$4.3834831618193604 \times 10^{-1}$	$4.3534267124176623 \times 10^{-1}$
16	$9.8107632446570958 \times 10^{-1}$	$9.5208962937681607 \times 10^{-1}$
20	$1.7042776030289366 \times 10^{0}$	$1.5057818246088250 \times 10^{0}$
25	$2.5674905431377995 \times 10^{0}$	$1.6432017599233490 \times 10^{0}$
30	$4.0560126128455938 \times 10^{0}$	$1.7809320553510379 \times 10^{0}$
36	$5.7109000664700984 \times 10^{0}$	$1.9262789521867196 \times 10^{0}$
42	$7.4825284953464246 \times 10^{0}$	$2.0820807241460830 \times 10^{0}$
49	$9.3385619211370852 \times 10^{0}$	$2.2421030188466875 \times 10^{0}$
56	$1.19081054947739435 \times 10^{1}$	$2.4876517018759325 \times 10^{0}$
64	$1.45559420698812616 \times 10^{1}$	$2.7461075372183124 \times 10^{0}$

If the inequation $\alpha_{\bar{m}} < \Theta_{\bar{P}_{\bar{m}}}$ or $\alpha_{\bar{m}} < \hat{\Theta}_{\bar{P}_{\bar{m}}}$ is not satisfied, then the smallest value of s is calculated such that $4^{-s}\alpha_{\bar{m}} < \Theta_{\bar{P}_{\bar{m}}}$ or $4^{-s}\alpha_{\bar{m}} < \hat{\Theta}_{\bar{P}_{\bar{m}}}$, respectively, for absolute or relative forward error, where

$$\alpha_{\bar{m}} = \max\left\{ \left\| \bar{A}^i \right\|^{1/i} : \ i = \bar{m}+1, \bar{m}+2, \cdots, 2\bar{m}+1 \right\}.$$

The values of $\Theta_{\bar{m}_k} \equiv \Theta_{\bar{P}_{\bar{m}_k}}$ and $\hat{\Theta}_{\bar{m}_k} \equiv \hat{\Theta}_{\bar{P}_{\bar{m}_k}}$ are listed in Table 2.

Table 2. Values of $\Theta_{\bar{m}_k}$ and $\hat{\Theta}_{\bar{m}_k}$, $\bar{m}_k \in \mathbb{M}$, for $E_{af}(\bar{P}_{\bar{m}_k}(\bar{A}))$ and $E_{rf}(\bar{P}_{\bar{m}_k}(\bar{A}))$, respectively, for Series (12).

$\bar{m}_k$	$\Theta_{\bar{m}_k}$	$\hat{\Theta}_{\bar{m}_k}$
1	$5.1619136514626776 \times 10^{-8}$	$5.1619135937310811 \times 10^{-8}$
2	$4.3077199749215582 \times 10^{-5}$	$4.3076912566764470 \times 10^{-5}$
4	$1.3213746092459254 \times 10^{-2}$	$1.3196809298927527 \times 10^{-2}$
6	$1.9214924629953856 \times 10^{-1}$	$1.8952324140391652 \times 10^{-1}$
9	$1.7498015129635465 \times 10^{0}$	$1.5605489459377038 \times 10^{0}$
12	$6.5920076891020321 \times 10^{0}$	$2.7025357197364501 \times 10^{0}$
16	$2.10870186062700462 \times 10^{1}$	$3.3425537406235706 \times 10^{0}$
20	$4.73520019672591133 \times 10^{1}$	$4.1166704209376803 \times 10^{0}$
25	$9.94413296329754246 \times 10^{1}$	$5.3203288339799650 \times 10^{0}$
30	$1.74869078212905435 \times 10^{2}$	$6.8352932849387500 \times 10^{0}$
36	$2.979204830753341753 \times 10^{2}$	$9.1455271414679480 \times 10^{0}$
42	$4.576519665452191248 \times 10^{2}$	$1.20985467228657093 \times 10^{1}$
49	$6.913637319746218282 \times 10^{2}$	$1.65236965866542853 \times 10^{1}$
56	$9.767604294039372235 \times 10^{2}$	$2.22163895347595428 \times 10^{1}$
64	$1.3667813478651733021 \times 10^{3}$	$3.05920634266714515 \times 10^{1}$

Finally, absolute and relative forward error series were also taken into account for the approximation of the matrix exponential in Algorithm 3 using matrix polynomial $\tilde{P}_m$. Table 3 collects the corresponding values of Θ_{m_k} and $\hat{\Theta}_{m_k}$ for $E_{af}(\tilde{P}_{m_k}(A))$ and $E_{rf}(\tilde{P}_{m_k}(A))$.

Table 3. Values of Θ_{m_k} and $\hat{\Theta}_{m_k}$, $m_k \in \mathbb{M}$, for the forward absolute and relative errors, $E_{af}(P_{m_k}(A))$ and $E_{rf}(P_{m_k}(A))$, of the exponential matrix.

m_k	Θ_{m_k}	$\hat{\Theta}_{m_k}$
1	$1.4901161156840223 \times 10^{-8}$	$1.4901161119832789 \times 10^{-8}$
2	$8.7334702258487179 \times 10^{-6}$	$8.7334575136353609 \times 10^{-6}$
4	$1.6783942982781048 \times 10^{-3}$	$1.6780188443217515 \times 10^{-3}$
6	$1.7764527083684662 \times 10^{-2}$	$1.7730821996540237 \times 10^{-2}$
9	$1.1483174747739708 \times 10^{-1}$	$1.1376892457878242 \times 10^{-1}$
12	$3.3521368782861483 \times 10^{-1}$	$3.2805420180372574 \times 10^{-1}$
16	$8.2460319163860885 \times 10^{-1}$	$7.9127401766002403 \times 10^{-1}$
20	$1.5041473223951629 \times 10^{0}$	$1.4150704475615321 \times 10^{0}$
25	$2.5585766884181380 \times 10^{0}$	$2.3536427669894273 \times 10^{0}$
30	$3.7810696269831392 \times 10^{0}$	$3.4118771725567707 \times 10^{0}$
36	$5.4064650937902918 \times 10^{0}$	$4.7855459552778310 \times 10^{0}$
42	$7.1556200904384877 \times 10^{0}$	$6.2345518738859917 \times 10^{0}$
49	$9.3073843996022152 \times 10^{0}$	$7.9882499230847923 \times 10^{0}$
56	$1.1545348315212191 \times 10^{1}$	$9.7882040407606592 \times 10^{0}$
64	$1.4179107337111319 \times 10^{1}$	$1.1884024795730356 \times 10^{1}$

Taking into account the precomputed values of Θ_{m_k} and $\hat{\Theta}_{m_k}$ of Tables 1–3, Algorithm 4 computes the most appropriate values of polynomial order m and scaling parameter s. In fact, Algorithm 4 is an improvement on [38]'s Algorithm 4, where it is explained in further detail. The main difference with respect to [38] is that the new code can be used

to determine the values of m and s independently of the nature of the error, covering relative, absolute, forward and backward errors. This is accommodated by means of Line 8 in Algorithm 4, which takes into account that the first non-zero term occupies position m_i, for the relative backward error series, or $m_i + 1$, for the absolute/relative forward or absolute backward error ones. Moreover, the new code is valid for matrix functions such as exponential, cosine and hyperbolic cosine by simply varying the corresponding values of Θ_m, always according to the type of error considered. For simplicity, in Algorithm 4, we use Θ to refer to Θ or $\hat{\Theta}$ of the corresponding polynomial, depending on the type of error considered, and we also use the following notation:

$$\alpha_i \equiv \alpha_{m_i}, \ \Theta_i \equiv \Theta_{m_i}.$$

Additionally, α_m is approximated as $\alpha_m \approx \|A^m\|^{1/m}$, as justified in [38]. In line 18, p_{m_i} is the highest-order coefficient of the approximating polynomial.

Algorithm 4: Given a matrix $A \in \mathbb{C}^{r\times r}$, a minimum order $m_{lower} \in \mathbb{M}$ and a maximum order $m_{upper} \in \mathbb{M}$, this algorithm provides an order $m \in \mathbb{M}$, $m_{lower} \leq m \leq m_{upper}$, a scaling factor s and the necessary powers of A to compute $\cosh(A)$ or $\exp(A)$

1 $A_1 = A; i = lower; f = 0$
2 **for** $j = 2$ **to** $\lceil \sqrt{m_i} \rceil$ **do**
3 $\quad A_j = A_{j-1}A$
4 **end**
5 **while** $f = 0$ **and** $i \leq upper$ **do**
6 $\quad v = \sqrt{m_i}; j = \lceil v \rceil$
7 $\quad$ **if** $j > v$ **then** $A_j = A_{j-1}A$
8 $\quad$ Compute $a_i \approx \left\|A^k\right\|$ from A^j, and maybe from A `/*` $k = m_i$ `(relative backward error) or` $k = m_i + 1$ `(absolute / relative forward error or absolute backward error) */`
9 $\quad \alpha_i = \sqrt[k]{a_i}$
10 $\quad$ **if** $\alpha_i < \Theta_i$ **then** $f = 1$
11 $\quad$ **else** $i = i + 1$
12 **end**
13 **if** $f = 1$ **then** $s = 0$
14 **else**
15 $\quad i = upper$
16 $\quad s = \max(0, \lceil f_s \log_2(\alpha_i/\Theta_i) \rceil)$ `/*` $f_s = 1$ `for Algorithms 1 and 3,` $f_s = 0.5$ `for Algorithm 2 */`
17 $\quad$ **while** $f = 0$ **do**
18 $\quad\quad$ **if** $s > 0$ **and** $|p_{m_i}| a_i r^{(1-s)m_i} < u$ **then** $s = s - 1$ `/*` $r = 2$ `(Algorithms 1 or 3) or` $r = 4$ `(Algorithm 2) */`
19 $\quad\quad$ **else** $f = 1$
20 $\quad$ **end**
21 **end**
22 $m = m_i$

3. Computational Experiments

In this section, a whole set of experiments carried out in order to compare the numerical and computational performance of the proposed algorithms are presented. For this purpose, the following codes, implemented in MATLAB programming language, were evaluated:

- `coshmber_ataf` and `coshmber_atrf`: They correspond to the coding of Algorithm 1, using the absolute or relative forward error, respectively. Polynomial degree m takes values from the set $\{25, 30, 36, 42, 49\}$.

- `coshmber_etaf` and `coshmber_etrf`: They are implementations of Algorithm 2, after considering the absolute or relative forward error. The values of $m \in \{16, 20, 25, 30\}$.
- `coshm_expmber_af` and `coshm_expmber_rf`: These functions include the implementation of Algorithm 3, where the absolute or relative forward error is correspondingly taken into account. Again, the values of $m \in \{25, 30, 36, 42, 49\}$.
- `coshm_expm`: This code also employs formula (14), but alternatively to the above ones (`coshm_expmber_af` and `coshm_expmber_rf`), the matrix exponential is computed by means of the code of MATLAB built-in function `expm`. Recall that function `expm` works out the matrix exponential combining the scaling and squaring technique with the Padé approximation [32,39].
- `funmcosh`: It consists of a short function that invokes the MATLAB built-in function `funm` to compute the matrix hyperbolic cosine. Function `funm` employs a Schur decomposition with reordering and blocking, and a block recurrence of Parlett [34]. It supports the matrix cosine, sine, hyperbolic cosine and hyperbolic sine. The derivatives of the matrix function to be approximated are also needed and computed.
- `funmcosh_nd_inf`: As in the previous case, it is just a simple code that calls function `funm_nd_inf`, implemented in [40], to calculate the hyperbolic cosine. More specifically, function `funm_nd_inf` is based on a multi-precision Schur–Parlett algorithm ([40] Algorithm 5.1) that does not require the matrix function derivatives. As blocking parameter δ is set to ∞ (no blocking), the whole Schur factor T is computed by [40]'s Algorithm 4.1.
  It is worth noting that, although function `funm_nd`, which is also implemented in [40] and which employs a value of $\delta = 0.1$, could have been used instead of function `funm_nd_inf`, the latter was finally chosen because it provided more accurate results in the different numerical experiments performed.

In the subsequent computational experiments, the following three sets of matrices were selected in an attempt to provide a test battery as numerically heterogeneous as possible. The *"exact"* value of the matrix hyperbolic cosine function was computed by means of MATLAB Symbolic Math Toolbox and the `vpa` (variable-precision floating-point arithmetic) function with 256 significant digits:

- Set 1: A total of 100 diagonalizable square complex matrices of order 128, generated as $A = V \cdot D \cdot V^{-1}$. V is an orthogonal matrix such that $V = H/\sqrt{n}$, with H being a Hadamard matrix and n its number of rows or columns, while D is a random diagonal matrix with complex eigenvalues. The 2-norm of the matrices varied from 0.1 to 350. The *"exact"* matrix hyperbolic cosine was computed as $\cosh(A) = V \cdot \cosh(D) \cdot V^T$ using the `vpa` function.
- Set 2: A total of 100 non-diagonalizable square complex matrices of size 128 and generated as $A = V \cdot J \cdot V^{-1}$. V is a matrix determined in exactly the same way as in the case of the previous set. However, J is a Jordan matrix with complex eigenvalues whose modules are less than five and with random algebraic multiplicity from 1 to 3. The 2-norm varied from 3.76 to 339.11. The matrix hyperbolic cosine was also *"exactly"* computed by means of the `vpa` function as $\cosh(A) = V \cdot \cosh(J) \cdot V^{-1}$.
- Set 3: A total of 72 square matrices of dimension 128, 52 of which are from Matrix Computation Toolbox (MCT) [41] and 20 from Eigtool MATLAB Package (EMP) [42]. Unfortunately, only 44 of these matrices (36 of MCT and 8 of EMP) could be successfully employed. The remaining matrices had to be excluded owing to the following reasons:
  - Their *"exact"* solution could not be computed: matrices 4, 5, 10, 16, 17, 18, 21, 25, 26, 35, 40, 42, 43, 44 and 49 from MCT and matrices 1, 5, 6, 7, 9 and 15 from EMP.
  - The relative error made by all the codes was too high due to their ill conditioning: matrix 2 from MCT and matrices 3 and 10 from EMP.
  - They were repetitive (already present in MCT): matrices 8, 11, 13 and 16 from EMP.

The *"exact"* calculation of the hyperbolic cosine of these matrices was performed in the following way:

- First, from initial matrix A and by means of the `eig` MATLAB function, a diagonal matrix D of eigenvalues and a matrix V whose columns were the corresponding eigenvectors were provided, such that $A = V \cdot D \cdot V^{-1}$. Thus, matrix $C_1 = V \cdot \cosh(D) \cdot V^{-1}$ was worked out.
- Second, matrix $C_2 = \cosh(A)$ was computed as the approximation to the hyperbolic cosine of matrix A through the scaling and squaring algorithm and Taylor polynomials using the `vpa` function.
- Finally, matrix C_1 was accepted as the *"exact"* solution in the calculation of the hyperbolic cosine of A if it was satisfied that

$$\frac{\|C_1 - C_2\|_2}{\|C_1\|_2} \leq u.$$

  Otherwise, matrix A was not part of the matrices of set 3.

All the executions were carried out on a Microsoft Windows 11 x64 PC equipped with an Intel Core i7-12700H processor and 32 GB of RAM, using MATLAB R2021b.

Henceforth, the normwise relative error made by each of the methods in the hyperbolic cosine computation for each test matrix was one of the key aspects to consider when comparing their goodness. This normwise relative error was obtained as follows:

$$\mathrm{Er(A)} = \frac{\|\cosh(A) - \widetilde{\cosh}(A)\|_2}{\|\cosh(A)\|_2},$$

where $\cosh(A)$ corresponds to the exact solution and $\widetilde{\cosh}(A)$ corresponds to the calculated one.

Experiment 1. *In this experiment, we compared the codes corresponding to Algorithms 1 and 2 (`coshmber_ataf`, `coshmber_atrf`, `coshmber_etaf` and `coshmber_etrf`) using the three matrix sets.*

Figure 1a,c,e show the normwise relative errors of the matrix hyperbolic cosine computed with those codes, with the solid line representing the value of $k_{\cosh}u$, where $k_{\cosh}$ is the estimated condition number of the matrix hyperbolic cosine function, obtained using function `funm_condest1` from Higham's *Matrix Function Toolbox* [5,43], and $u = 2^{-53}$ is the unit roundoff error. Thus, the solid line is an indication of the expected relative error. The results show the stability of the methods, especially when applied to the matrices of sets 1 and 2, for which the relative error remained below the solid line. Although the results were more irregular for the third matrix set, the error values were small in all cases. We can also see in Figure 1e that `coshmber_etaf` produced the highest error for a considerable number of matrices of set 3.

It should be noted that nine matrices of the third set are excluded from Figure 1e because their estimated condition number $k_{\cosh}$ was too high. The same is also performed in Figure 4e, which is referenced later. These matrices were number 6, 7, 12, 15, 23, 36, 39, 50 and 51 from MCT.

Figure 1b,d,f present *performance profiles* comparing the accuracy of the codes on the different matrix sets. For a given value α on the x-axis, the value of p on the y-axis is the proportion of matrices for which the considered code had a relative error lower than or equal to α times the smallest relative error of all the codes for the matrix. Performance plots are explained in detail in [5] (section 10.5) and are quite accepted. The different accuracy of the methods on matrices of the third set was very apparent, as can be seen in Figure 1f, where code `coshmber_etrf` clearly outperformed the other methods, while as anticipated above, `coshmber_etaf` was the least accurate alternative. The remaining two codes were very similar in terms of accuracy. For sets 1 and 2, the differences were less important, although we can see that `coshmber_etrf` was also on top for lower values of α for the

second set and for higher values of α for the first set, confirming it as the most accurate option. It can also be seen that `coshmber_etaf` was the least accurate option for the first set and was also below the other codes for high values of α for the second set.

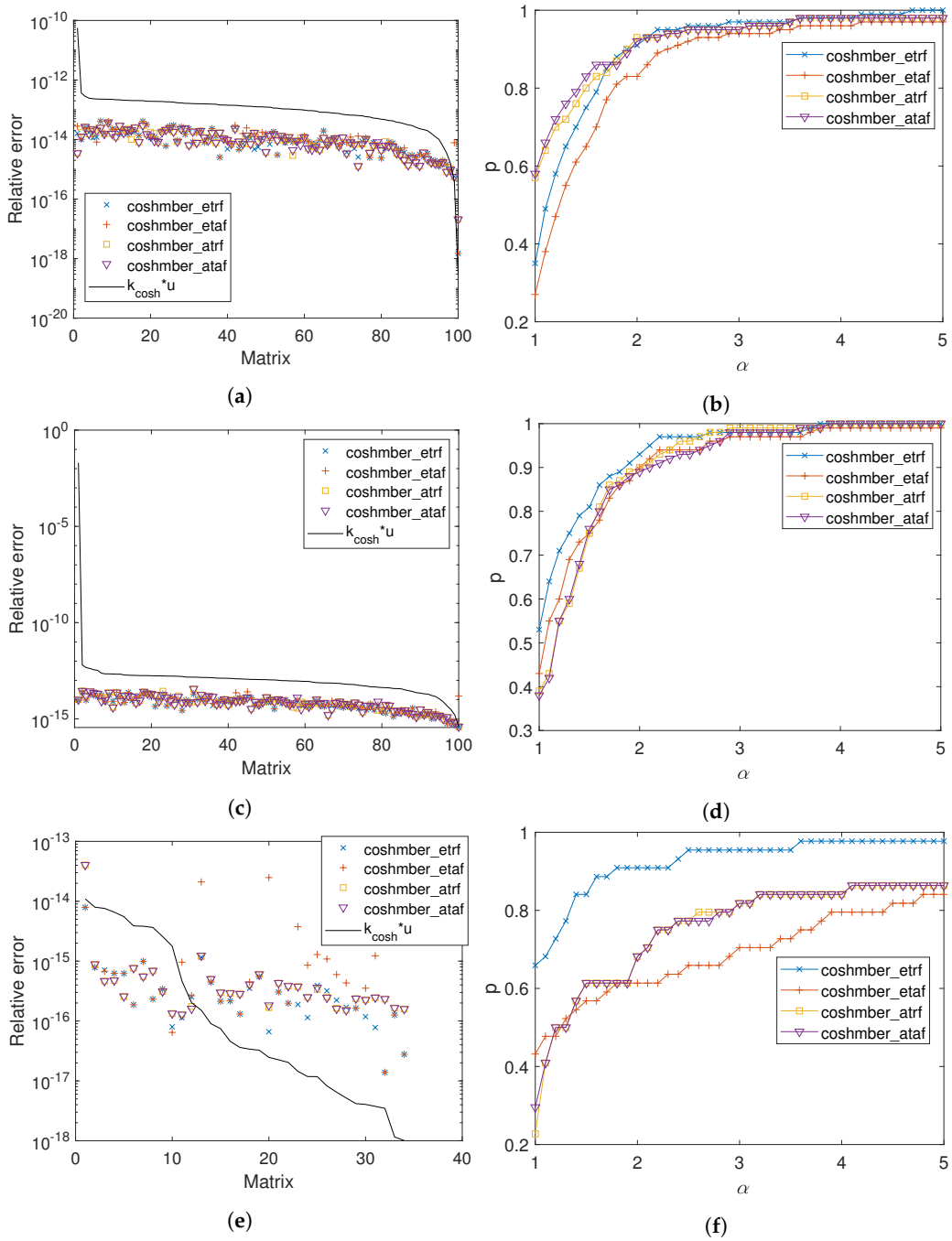

Figure 1. Experiment 1: Normwise relative errors for matrix sets 1 (**a**), 2 (**c**) and 3 (**e**) and performance plots for the same sets (**b**,**d**,**f**).

Figure 2 shows the proportion of matrices, in each set, for which each code provided the lowest/highest error. It confirms that `coshmber_etrf` was the most accurate option, especially taking into account sets 3 and 2, while `coshmber_etaf` was the least accurate option for the first set. It also shows that `coshmber_atrf` and `coshmber_ataf` were very similar in terms of accuracy, although the latter seemed to be slightly better.

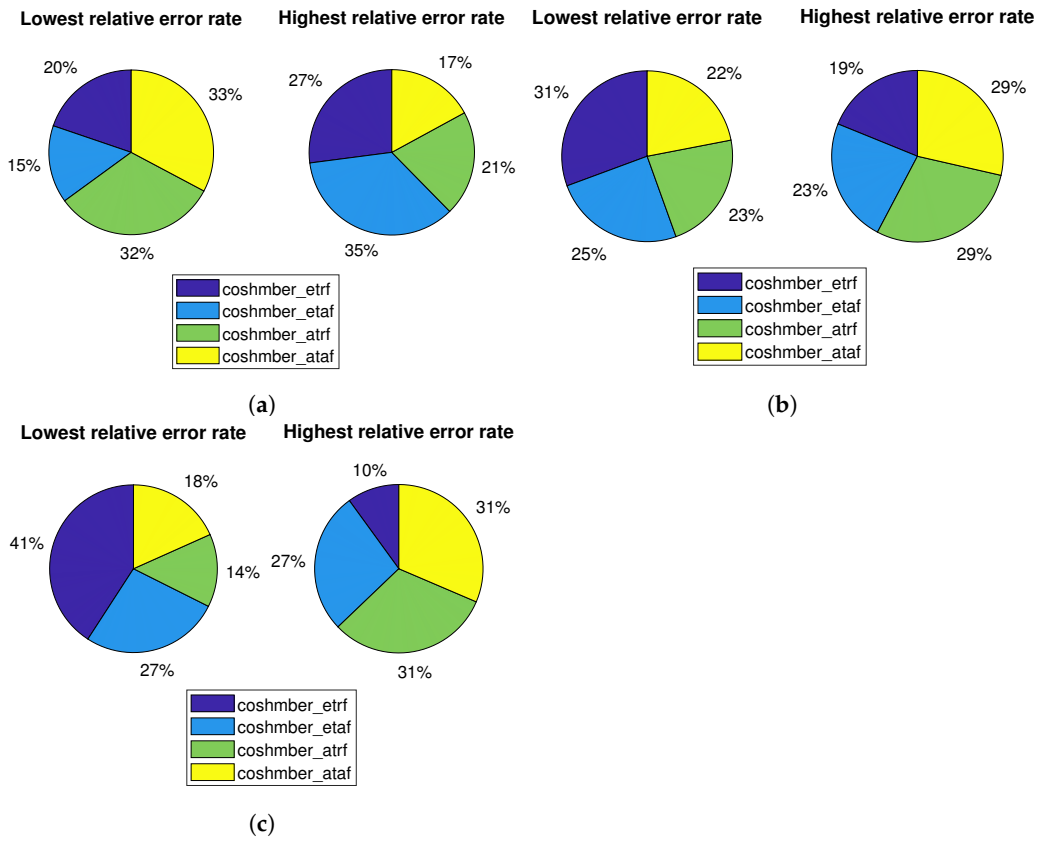

Figure 2. Experiment 1: Proportion of matrices for which each code provided the lowest/highest error for matrix sets 1 (**a**), 2 (**b**) and 3 (**c**).

Table 4 considers the computational costs, in terms of number of matrix products for all the matrices within a set, of the different codes in experiment 1. We can see that the codes based on Algorithm 2, which use polynomials containing even-order terms only, required a lower number of products.

Table 4. Experiment 1: Number of matrix products for all the matrices in a set for each code.

Code	Set 1	Set 2	Set 3
`coshmber_etrf`	1306	1303	424
`coshmber_etaf`	1276	1273	388
`coshmber_atrf`	1638	1637	563
`coshmber_ataf`	1622	1623	527

As a conclusion of experiment 1, `coshmber_etrf` was identified as the most accurate code and also as one of the most efficient. It was certainly much more accurate than `coshmber_etaf`, which is also based on Algorithm 2. With respect to the two the codes based on Algorithm 1, they were very similar in terms of accuracy and computational cost, although `coshmber_ataf` seemed to be slightly better in both respects.

For completeness, Figure 3 presents box plots of the values of parameters m and s selected by Algorithm 4 for the tests of experiment 1. In each box, the central mark indicates the median, and the bottom and top edges of the box indicate the 25th percentile (q_1) and the 75th percentile (q_3), respectively. The whiskers extend to the most extreme data points in the interval of $[q_1 - 1.5(q_3 - q_1),\ q_3 + 1.5(q_3 - q_1)]$, while the values outside of the interval were considered outliers and are marked with a '+' symbol. Figure 3 shows that the order m of the polynomials used by `coshmber_etrf` and `coshmber_etaf`, corresponding to expression (12), was lower than that of the other two codes, corresponding to expression (10). We can also see that for sets 1 and 2, m took the same value within a given method

almost always ($m = 30$ for `coshmber_etrf` and `coshmber_etaf` and $m = 49$ for the other two methods). The values of s were similar for all the methods, although slightly lower for `coshmber_etrf` and `coshmber_etaf`, especially in set 2.

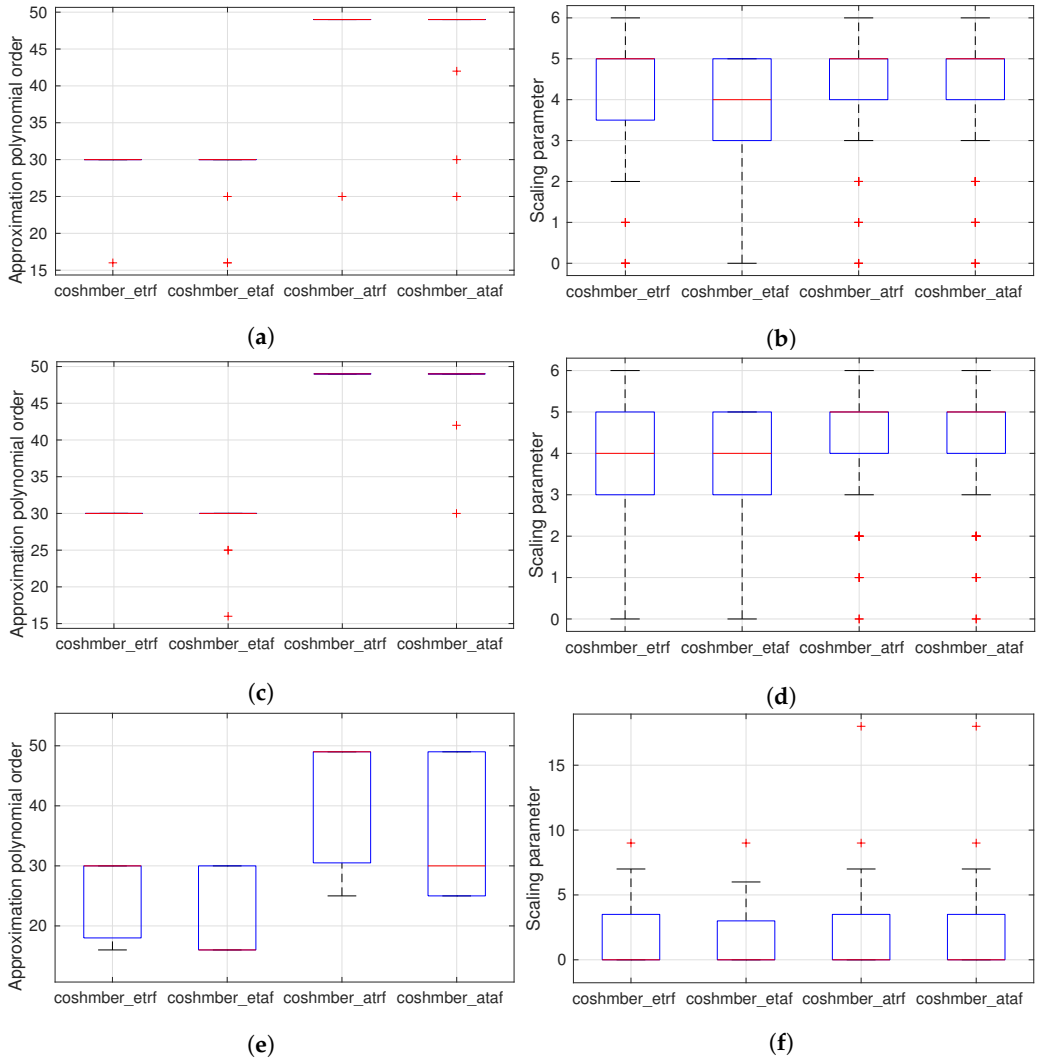

Figure 3. Experiment 1: Values of polynomial degree m for matrix sets 1 (**a**), 2 (**c**) and 3 (**e**) and scaling parameters s for the same sets (**b**,**d**,**f**).

Experiment 2. *In this experiment, we took the best code identified in the previous experiment (`coshmber_etrf`) together with the best code based on Algorithm 1 (`coshmber_ataf`) and compared them with functions `coshm_expmber_af` and `coshm_expmber_rf`, corresponding to Algorithm 3, and with other options based on state-of-the-art approaches, such as functions `coshm_expm`, `funmcosh_nd_inf` and `funmcosh`. Matrix 4 from EMP was excluded from the third matrix set in this experiment, because `funmcosh` could not compute its hyperbolic cosine.*

Similarly to Experiment 1, Figure 4a,c,e show the normwise relative errors of the different codes, together with the value of $k_{\cosh}u$ given by the solid line. We can see that `funmcosh` and `funmcosh_nd_inf` produced considerably larger errors than the other codes. This was especially true for some matrices of the third set, where the result produced by `funmcosh` was very inaccurate. The other codes presented good stability, with errors below the solid line for sets 1 and 2 and not far from that line for set 3.

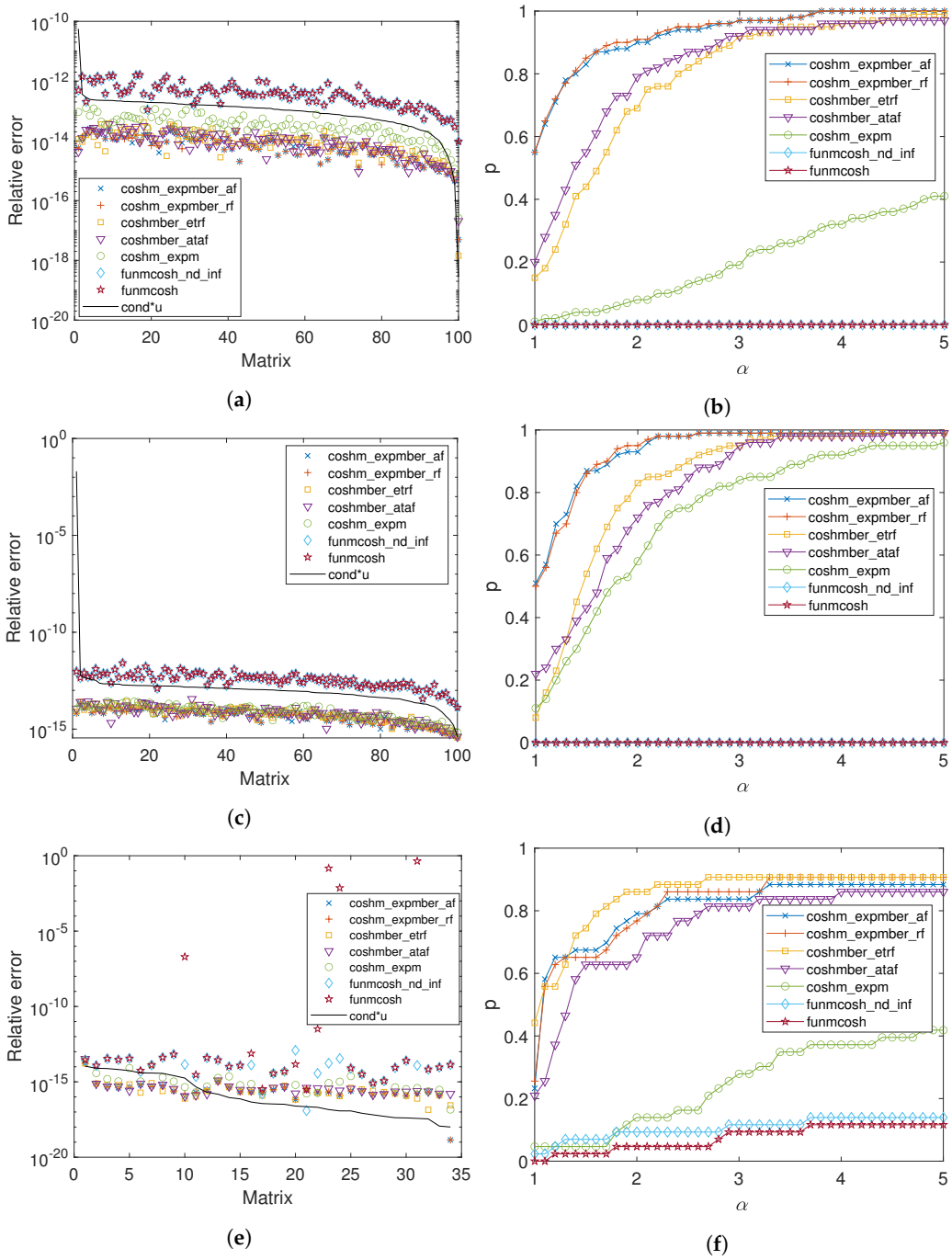

Figure 4. Experiment 2: Normwise relative errors for matrix sets 1 (**a**), 2 (**c**) and 3 (**e**) and performance plots for the same sets (**b**,**d**,**f**).

In Figure 4b,d,f, we can see the performance plots for the different codes. It is clear that `funmcosh` and `funmcosh_nd_inf` were the options with the worst performance for any matrix set, as indicated by the lower values of the profile in the three figures. The next worst one was `coshm_expm`. We can also see that `coshm_expmber_af` and `coshm_expmber_rf` were the best options for matrix sets 1 and 2, while for the third set, `coshmber_etrf` was the best option, followed by the two previous codes.

Figure 5 shows the proportion of matrices in each set for which each code provided the lowest/highest error. It confirms that `coshm_expmber_af` and `coshm_expmber_rf` were the best options for matrix sets 1 and 2, while `coshmber_etrf` was the best option for the third matrix set. It also shows that `funmcosh` and `funmcosh_nd_inf` performed the worst.

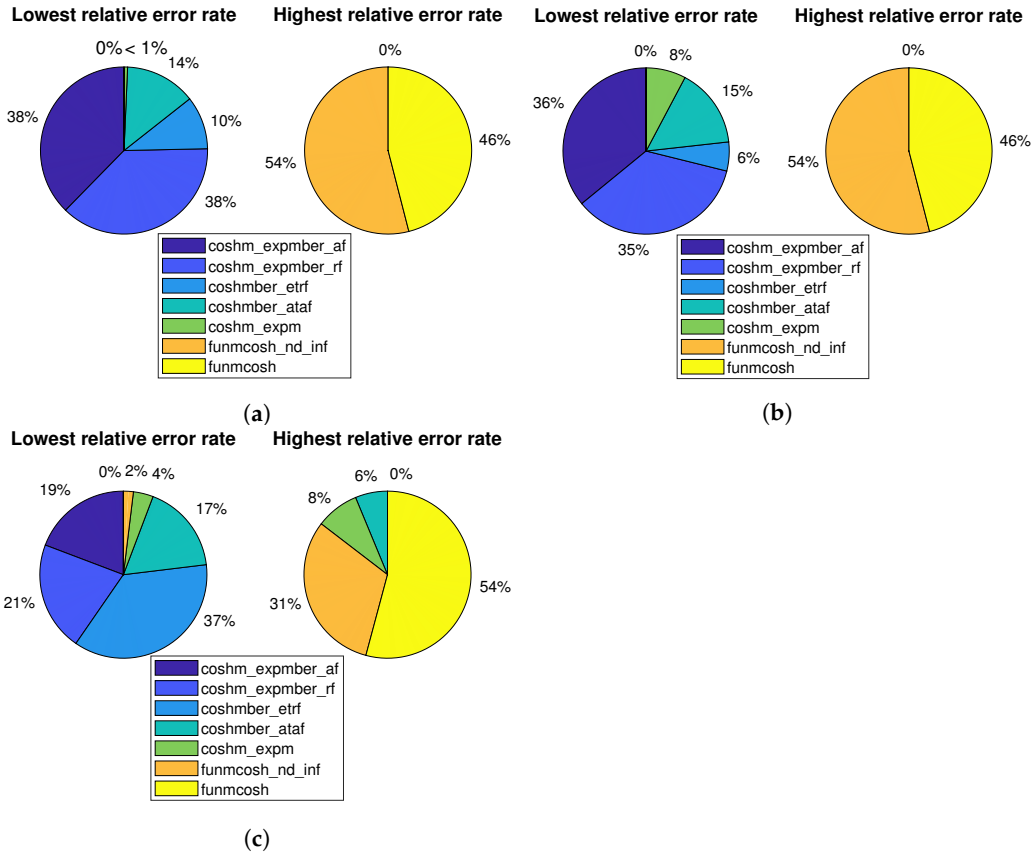

Figure 5. Experiment 2: Proportion of matrices for which each code provided the lowest/highest error for matrix sets 1 (**a**), 2 (**b**) and 3 (**c**).

Table 5 considers the computational costs, in terms of number of matrix products for all the matrices in each set, of the different codes in this experiment. It should be clarified that, although function `coshm_expm` uses the code of function `expm`, some modifications were performed to avoid repeating computations common to the exponentials of A and $-A$. In particular, the Schur decomposition, and the computation of the scaling parameters and the order of the Padé approximant were performed only once.

Table 5. Experiment 2: Number of matrix products for all the matrices in a set for each code.

Code	Set 1	Set 2	Set 3
`coshm_expmber_af`	2649	2651	776
`coshm_expmber_rf`	2697	2693	790
`coshmber_etrf`	1306	1303	411
`coshmber_ataf`	1622	1623	497
`coshm_expm`	2894	2891	645
`funmcosh_nd_inf`	1433	1433	616
`funmcosh`	1400–2233	1400–2233	602–917

Function `funmcosh` does not perform matrix products, being based instead on the Schur–Parlett algorithm with a computational cost between $28n^3$ and $\frac{1}{3}n^4$ flops [34]. Similarly and according to [40], the cost of function `funm_nd_inf`, and consequently of function `funmcosh_nd_inf`, consists of $28n^3$ flops using the desired precision plus $\frac{2}{3}n^3$ flops using higher precision. In our case, where $n = 128$, this can be respectively translated to a number of matrix products between 14 and 22.33 for `funmcosh` and 14.33 for `funmcosh_nd_inf`, assuming that the cost of a matrix product is $2n^3$ flops. As we can see, `coshmber_etrf` was the code with the lowest computational cost, while `coshm_expm` and the two codes based on Algorithm 3 were the most demanding ones, requiring approximately twice the number

of matrix products of `coshmber_etrf`. This comes from the fact that the codes based on Algorithm 3 and function `coshm_expm` need to evaluate the matrix exponential twice.

Figure 6 presents box plots for order m of the polynomials and scaling parameter s selected by Algorithm 4 in each of the codes of the experiment, except for `funmcosh` and `funmcosh_nd_inf`, for which the parameters were not available. We can see that `coshm_expm` generally used lower polynomial orders and higher scaling parameters than the other codes. Among those other codes, `coshm_expm` used lower orders, while the value of s was similar for all them.

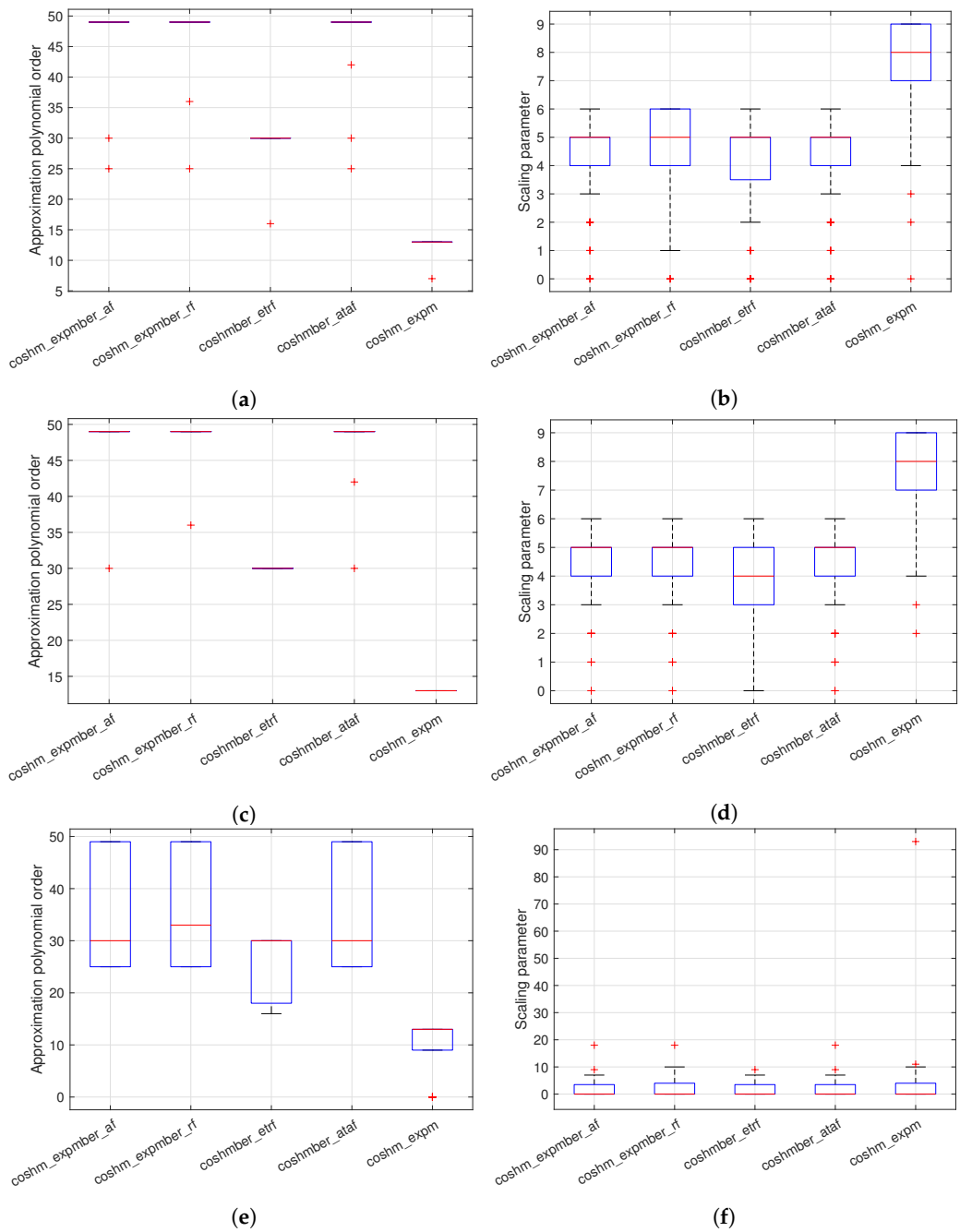

Figure 6. Experiment 2: Values of polynomial degree m for matrix sets 1 (**a**), 2 (**c**) and 3 (**e**) and scaling parameters s for the same sets (**b**,**d**,**f**).

In summary, we can conclude that codes `coshm_expmber_rf`, `coshm_expmber_af` and `coshmber_etrf` are the most accurate ones, clearly outperforming options based on existing

approaches, such as `funmcosh`, `funmcosh_nd_inf` or `coshm_expm`. Code `coshmber_etrf` is also the best option from the point of view of computational cost.

4. Conclusions

In this work, we propose and analyze three different algorithms to approximate the matrix hyperbolic cosine by means of Bernoulli polynomials. Two of the proposed approximations come from different Bernoulli series expansions of the matrix hyperbolic cosine. One of them consists of both even- and odd-order terms, while the other contains only even-order terms. The third proposed approximation comes from the approximation of the matrix exponential using Bernoulli matrix polynomials.

The selection of polynomial order m and scale parameter s is performed by means of Algorithm 4, which is an extension of [38]'s Algorithm 4 to consider any type of error, either absolute or relative, forward or backward. Algorithm 4 uses precomputed values of Θ_m as inputs. These values are provided in this paper for the different polynomial approximations and for absolute and relative forward error types.

By combining the three different approximation algorithms with the two error types (absolute or relative), we obtained six different computing codes to approximate the matrix hyperbolic cosine. These codes were compared using a comprehensive testbed of matrices with different characteristics. We also compared the codes with reference methods based on current state-of-the-art approaches.

From the point of view of accuracy, codes `coshm_expmber_rf` and `coshm_expmber_af` (corresponding to Algorithm 3 with relative and absolute forward error, respectively), and `coshmber_etrf` (based on Algorithm 2 with relative error) emerged as the best options. All of them clearly outperformed the alternatives, such as `funmcosh`, `funmcosh_nd_inf` and `coshm_expm`, based on state-of-the-art methods. In terms of computational cost, the number of matrix products of codes `coshm_expmber_rf` and `coshm_expmber_af` approximately doubled that of the less demanding option, `coshmber_etrf`. Thus, `coshmber_etrf` is a good option for applications where the computational cost is an important consideration.

Author Contributions: Conceptualization, E.D. and J.I.; methodology, J.M.A., J.I., E.D. and F.A.; software, J.M.A. and J.I.; validation, J.M.A. and F.A.; formal analysis, E.D.; writing—original draft preparation, J.M.A. and F.A.; writing—review and editing, J.I. and E.D.; visualization, J.M.A. and F.A.; supervision, E.D.; project administration, J.M.A.; funding acquisition, J.M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Vicerrectorado de Investigación de la Universitat Politècnica de València (PAID-11-21).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Remark A1. *We proof here expression (3). Firstly, we start from formula (24.5.3) of ([29] p. 591).*

$$\sum_{k=0}^{n-1} \binom{n}{k} \mathcal{B}_k = 0, n \geq 2. \tag{A1}$$

If we replace the value of $n-1$ with m in (A1), we obtain, for $m \geq 1$,

$$\sum_{k=0}^{m} \binom{m+1}{k} \mathcal{B}_k = 0. \tag{A2}$$

By developing (A2), we have

$$\sum_{k=0}^{m-1} \binom{m+1}{k} \mathcal{B}_k + \binom{m+1}{m} \mathcal{B}_m = 0,$$

and so

$$(m+1)\mathcal{B}_m = -\sum_{k=0}^{m-1}\binom{m+1}{k}\mathcal{B}_k.$$

Therefore, for $m \geq 1$,

$$\begin{aligned}
\mathcal{B}_m &= -\sum_{k=0}^{m-1}\binom{m+1}{k}\frac{\mathcal{B}_k}{m+1} \\
&= -\sum_{k=0}^{m-1}\frac{(m+1)!}{k!(m+1-k)!}\frac{\mathcal{B}_k}{m+1} \\
&= -\sum_{k=0}^{m-1}\frac{m!}{k!(m-k)!}\frac{\mathcal{B}_k}{m+1-k} \\
&= -\sum_{k=0}^{m-1}\binom{m}{k}\frac{\mathcal{B}_k}{m+1-k}.
\end{aligned}$$

By replacing m with n, we finally obtain formula (3).

Remark A2. *We proof here expression (8). Using (5) with $t = 1$ and $t = -1$, we have*

$$\begin{aligned}
\cosh(A) &= \frac{1}{2}\left(e^A + e^{-A}\right) \\
&= \frac{1}{2}(e-1)\sum_{n\geq 0}\frac{B_n(A)}{n!} + \frac{1}{2}\left(1-e^{-1}\right)\sum_{n\geq 0}\frac{B_n(A)(-1)^n}{n!} \\
&= \sum_{n\geq 0}\left(\frac{(e-1)+\left(1-e^{-1}\right)(-1)^n}{2}\right)\frac{B_n(A)}{n!}.
\end{aligned}$$

By separating even indices from odd indices, we obtain

$$\begin{aligned}
\cosh(A) &= \sum_{n\geq 0}\left(\frac{(e-1)+\left(1-e^{-1}\right)}{2}\right)\frac{B_{2n}(A)}{(2n)!} + \sum_{n\geq 0}\left(\frac{(e-1)-\left(1-e^{-1}\right)}{2}\right)\frac{B_{2n+1}(A)}{(2n+1)!} \\
&= \sum_{n\geq 0}\left(\frac{e-e^{-1}}{2}\right)\frac{B_{2n}(A)}{(2n)!} + \sum_{n\geq 0}\left(\frac{e+e^{-1}-2}{2}\right)\frac{B_{2n+1}(A)}{(2n+1)!} \\
&= \sinh(1)\sum_{n\geq 0}\frac{B_{2n}(A)}{(2n)!} + (\cosh(1)-1)\sum_{n\geq 0}\frac{B_{2n+1}(A)}{(2n+1)!},
\end{aligned}$$

which corresponds to expression (8).

Remark A3. *We now proof expression (9). Equation (5) can be written as*

$$\sum_{n\geq 0}\frac{B_n(A)t^n}{n!} = \frac{te^{At}}{e^t-1}\,,\ |t| < 2\pi, \tag{A3}$$

and, by replacing t with $-t$,

$$\sum_{n\geq 0}\frac{B_n(A)(-1)^n t^n}{n!} = \frac{-te^{-At}}{e^{-t}-1}\,,\ |t| < 2\pi. \tag{A4}$$

By adding (A3) and (A4), we have

$$\sum_{n\geq 0}\frac{B_n(A)t^n}{n!} + \sum_{n\geq 0}\frac{B_n(A)(-1)^n t^n}{n!} = \frac{te^{At}}{e^t-1} + \frac{-te^{-At}}{e^{-t}-1}\,,\ |t| < 2\pi. \tag{A5}$$

The left side of (A5) is equal to $2\sum_{n\geq 0}\frac{B_{2n}(A)t^{2n}}{(2n)!}$. *Thus,*

$$\sum_{n\geq 0}\frac{B_{2n}(A)t^{2n}}{(2n)!}=\frac{t}{2}\left(\frac{e^{At}}{e^{t}-1}-\frac{e^{-At}}{e^{-t}-1}\right),\ |t|<2\pi. \tag{A6}$$

We now rewrite the expression $\frac{t}{2}\left(\frac{\cosh\left((2A-I)(t/2)\right)}{\sinh\left(t/2\right)}\right)$ *in the following way:*

$$\begin{aligned}
\frac{t}{2}\left(\frac{\cosh\left((2A-I)(t/2)\right)}{\sinh\left(t/2\right)}\right) &= \frac{t}{2}\frac{2}{e^{t/2}-e^{-t/2}}\left(\frac{e^{(2A-I)(t/2)}+e^{-(2A-I)(t/2)}}{2}\right)\\
&= \frac{t/2}{e^{t/2}-e^{-t/2}}\left(e^{At}e^{-t/2}+e^{-At}e^{t/2}\right)\\
&= \frac{t/2}{e^{t/2}-e^{-t/2}}\left(e^{At}e^{-t/2}+e^{-At}e^{t/2}\right)\frac{e^{t/2}-e^{-t/2}}{e^{t/2}-e^{-t/2}}\\
&= \frac{t/2}{e^{t}-2+e^{-t}}\left(e^{At}-e^{At}e^{-t}+e^{-At}e^{t}-e^{-At}\right)\\
&= \frac{t/2}{2-e^{t}-e^{-t}}\left(e^{At}e^{-t}-e^{At}-e^{-At}e^{t}+e^{-At}\right)\\
&= \frac{t/2}{(e^{t}-1)(e^{-t}-1)}\left(e^{At}(e^{-t}-1)-e^{-At}(e^{t}-1)\right)\\
&= \frac{t}{2}\left(\frac{e^{At}}{e^{t}-1}-\frac{e^{-At}}{e^{-t}-1}\right).
\end{aligned}$$

By substituting into (A6)

$$\sum_{n\geq 0}\frac{B_{2n}(A)t^{2n}}{(2n)!}=\frac{t}{2}\left(\frac{\cosh\left((2A-I)(t/2)\right)}{\sinh\left(t/2\right)}\right),\ |t|<2\pi,$$

and rearranging the terms, we obtain

$$\cosh\left((2A-I)(t/2)\right)=\frac{\sinh\left(t/2\right)}{t/2}\sum_{n\geq 0}\frac{B_{2n}(A)t^{2n}}{(2n)!},\ |t|<2\pi.$$

By defining variables $u=t/2$ *and* $C=2A-I$, *we have*

$$\cosh\left(Cu\right)=\frac{\sinh u}{u}\sum_{n\geq 0}\frac{B_{2n}(\frac{1}{2}(C+I))(2u)^{2n}}{(2n)!},\ |u|<\pi,$$

and by taking $u=1$, *we obtain*

$$\cosh C=\sinh\left(1\right)\sum_{n\geq 0}\frac{2^{2n}B_{2n}(\frac{1}{2}(C+I))}{(2n)!},$$

which corresponds to expression (9).

References

1. Druskin, V.; Mamonov, A.V.; Zaslavsky, M. Multiscale S-fraction reduced-order models for massive wavefield simulations. *Multiscale Model. Simul.* **2017**, *15*, 445–475. [CrossRef]
2. Frommer, A.; Simoncini, V. Matrix Functions. In *Model Order Reduction: Theory, Research Aspects and Applications. Mathematics in Industry. The European Consortium for Mathematics in Industry*; Springer: Berlin/Heidelberg, Germany, 2008; Volume 13, pp. 275–303.
3. May, V.; Keller, Y.; Sharon, N.; Shkolnisky, Y. An algorithm for improving non-local means operators via low-rank approximation. *IEEE Trans. Image Process.* **2016**, *25*, 1340–1353. [CrossRef]

4. Levie, R.; Monti, F.; Bresson, X.; Bronstein, M.M. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Trans. Signal Process.* **2018**, *67*, 97–109. [CrossRef]
5. Higham, N.J. *Functions of Matrices: Theory and Computation*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2008.
6. Defez, E.; Sastre, J.; Ibánez, J.; Peinado, J.; Tung, M.M. On the computation of the hyperbolic sine and cosine matrix functions. *Model. Eng. Hum. Behav.* **2013**, *2013*, 46.
7. Defez, E.; Sastre, J.; Ibáñez, J.; Peinado, J.; Tung, M.M. A method to approximate the hyperbolic sine of a matrix. *Int. J. Complex Syst. Sci.* **2014**, *4*, 41–45.
8. Defez, E.; Sastre, J.; Ibáñez, J.; Ruiz, P. Computing hyperbolic matrix functions using orthogonal matrix polynomials. In *Progress in Industrial Mathematics at ECMI 2012. Mathematics in Industry. The European Consortium for Mathematics in Industry*; Springer International Publishing: Cham, Switzerland, 2014; Volume 19, pp. 403–407.
9. Defez, E.; Sastre, J.; Ibáñez, J.; Peinado, J. Solving engineering models using hyperbolic matrix functions. *Appl. Math. Model.* **2016**, *40*, 2837–2844. [CrossRef]
10. Defez, E.; Ibáñez, J.; Peinado, J.; Alonso-Jordá, P.; Alonso, J.M. New Hermite series expansion for computing the matrix hyperbolic cosine. *J. Comput. Appl. Math.* **2022**, *408*, 114084. [CrossRef]
11. Bahşi, M.; Solak, S. On the hyperbolic Fibonacci matrix functions. *Twms J. Appl. Eng. Math.* **2018**, *8*, 454–465.
12. Bahşi, M.; Mersin, E.Ö. On the hyperbolic Horadam matrix functions. *Hacet. J. Math. Stat.* **2022**, *51*, 1550–1562. [CrossRef]
13. Estrada, E.; Higham, D.J.; Hatano, N. Communicability and multipartite structures in complex networks at negative absolute temperatures. *Phys. Rev. E* **2008**, *78*, 026102. [CrossRef]
14. Tseng, C.C.; Lee, S.L. Identification of Station Importance of Taipei Metro Network Using Subgraph Centrality. In Proceedings of the 2021 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), Hualien, Taiwan, 16–19 November 2021; pp. 1–2.
15. Jódar, L.; Navarro, E.; Martin, J.A. Exact and analytic-numerical solutions of strongly coupled mixed diffusion problems. *Proc. Edinb. Math. Soc.* **2000**, *43*, 269–293. [CrossRef]
16. Jódar, L.; Navarro, E.; Posso, A.; Casabán, M. Constructive solution of strongly coupled continuous hyperbolic mixed problems. *Appl. Numer. Math.* **2003**, *47*, 477–492. [CrossRef]
17. Aprahamian, M.; Higham, N.J. Matrix inverse trigonometric and inverse hyperbolic functions: Theory and algorithms. *SIAM J. Matrix Anal. Appl.* **2016**, *37*, 1453–1477. [CrossRef]
18. Higham, N.J.; Kandolf, P. Computing the action of trigonometric and hyperbolic matrix functions. *SIAM J. Sci. Comput.* **2017**, *39*, A613–A627. [CrossRef]
19. Al-Mohy, A.H. A Truncated Taylor Series Algorithm for Computing the Action of Trigonometric and Hyperbolic Matrix Functions. *SIAM J. Sci. Comput.* **2018**, *40*, A1696–A1713. [CrossRef]
20. Ibáñez, J.; Alonso, J.M.; Sastre, J.; Defez, E.; Alonso-Jordá, P. Advances in the Approximation of the Matrix Hyperbolic Tangent. *Mathematics* **2021**, *9*, 1219. [CrossRef]
21. Efimov, G.V.; Von Waldenfels, W.; Wehrse, R. Analytical solution of the non-discretized radiative transfer equation for a slab of finite optical depth. *J. Quant. Spectrosc. Radiat. Transf.* **1995**, *53*, 59–74. [CrossRef]
22. Lehtinen, A. Analytical Treatment of Heat Sinks Cooled by Forced Convection. Ph.D. Thesis, Tampere University of Technology, Tampere, Finland, 2005.
23. Lampio, K. Optimization of Fin Arrays Cooled by Forced or Natural Convection. Ph.D. Thesis, Tampere University of Technology, Tampere, Finland, 2018.
24. Hilscher, R.; Zemánek, P. Trigonometric and hyperbolic systems on time scales. *Dyn. Syst. Appl.* **2009**, *18*, 483.
25. Zemánek, P. New Results in Theory of Symplectic Systems on Time Scales. Ph.D. Thesis, Masarykova Univerzita, Brno, Czech Republic, 2011.
26. Estrada, E.; Silver, G. Accounting for the role of long walks on networks via a new matrix function. *J. Math. Anal. Appl.* **2017**, *449*, 1581–1600. [CrossRef]
27. Cieśliński, J.L. Locally exact modifications of numerical schemes. *Comput. Math. Appl.* **2013**, *65*, 1920–1938. [CrossRef]
28. Cieśliński, J.L.; Kobus, A. Locally Exact Integrators for the Duffing Equation. *Mathematics* **2020**, *8*, 231. [CrossRef]
29. Olver, F.W.; Lozier, D.W.; Boisvert, R.F.; Clark, C.W. *NIST Handbook of Mathematical Functions Hardback and CD-ROM*; Cambridge University Press: Cambridge, UK, 2010.
30. Defez, E.; Ibáñez, J.; Alonso-Jordá, P.; Alonso, J.M.; Peinado, J. On Bernoulli matrix polynomials and matrix exponential approximation. *J. Comput. Appl. Math.* **2022**, *404*, 113207. [CrossRef]
31. Higham, N.J. *The Scaling and Squaring Method for the Matrix Exponential Revisited*; Technical Report 452; Manchester Centre for Computational Mathematics: Manchester, UK, 2004.
32. Al-Mohy, A.H.; Higham, N.J. A New Scaling and Squaring Algorithm for the Matrix Exponential. *SIAM J. Matrix Anal. Appl.* **2009**, *31*, 970–989. [CrossRef]
33. Higham, N.J.; Hopkins, E. *A Catalogue of Software for Matrix Functions*; Version 3.0; MIMS EPrint 2020.7; Manchester Institute for Mathematical Sciences, The University of Manchester: Manchester, UK, 2020.
34. Davies, P.I.; Higham, N.J. A Schur–Parlett Algorithm for Computing Matrix Functions. *SIAM J. Matrix Anal. Appl.* **2003**, *25*, 464–485. [CrossRef]

35. Paterson, M.S.; Stockmeyer, L.J. On the Number of Nonscalar Multiplications Necessary to Evaluate Polynomials. *SIAM J. Comput.* **1973**, *2*, 60–66. [CrossRef]
36. Sastre, J. Efficient evaluation of matrix polynomials. *Linear Algebra Appl.* **2018**, *539*, 229–250. [CrossRef]
37. Sastre, J.; Ibáñez, J.; Ruiz, P.; Defez, E. Accurate and efficient matrix exponential computation. *Int. J. Comput. Math.* **2013**, *91*, 97–112. [CrossRef]
38. Defez, E.; Ibáñez, J.; Alonso, J.M.; Alonso-Jordá, P. On Bernoulli series approximation for the matrix cosine. *Math. Methods Appl. Sci.* **2022**, *45*, 3239–3253. [CrossRef]
39. Higham, N.J. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.* **2005**, *26*, 1179–1193. [CrossRef]
40. Higham, N.J.; Liu, X. A multiprecision derivative-free Schur– algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.* **2021**, *42*, 1401–1422. [CrossRef]
41. Higham, N.J. The Matrix Computation Toolbox. 2002. Available online: http://www.ma.man.ac.uk/~higham/mctoolbox (accessed on 16 January 2023).
42. Wright, T.G. Eigtool, Version 2.1. 2009. Available online: http://www.comlab.ox.ac.uk/pseudospectra/eigtool (accessed on 16 January 2023).
43. Higham, N.J. The Matrix Function Toolbox. Available online: http://www.ma.man.ac.uk/~higham/mftoolbox (accessed on 16 January 2023).